SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 8, 49–75 (1996)

Practice

From Software Configuration to Application Management—Improving the Maturity of the Maintenance of Embedded Software

JORMA TARAMAA, VEIKKO SEPPÄNEN AND MINNA MÄKÄRÄINEN
VTT Electronics, P.O. Box 1100, FIN-90571, Oulu, Finland

SUMMARY

The electronic and automation industries develop and maintain software embedded in computer-controlled products. Configuration management is one of the basic activities that must be performed in order to control product level changes for maintaining product integrity. Practitioners have traditionally seen version control as a sufficient solution to the configuration management problem. However, increasingly complex products have forced them to consider the problem more comprehensively. Flexible delivery of products based on automated change control is a key activity in application management which is a new challenge for traditional configuration management practices.

Companies developing embedded software may operate with a rather low level of maintenance process maturity. The assessment of software process maturity is based on various factors. Configuration Management (CM) is one of the basic goals at the 'repeatable' level of the Capability Maturity Model (CMM). The higher the maturity levels that are sought, the more integrated the CM environment used has to be. The definition and implementation of change control is one of the basic problems to be solved when seeking advanced CM practices. Change control can be regarded as a bridge from software maintenance to configuration management.

Although CM is regarded as a known technology, the maintenance of embedded computer systems sets new types of requirements. One of the major differences between embedded and legacy systems has been the secondary role of software in several embedded computer systems. In this paper, we describe an incremental approach advancing from CM to application management which was developed in co-operation with industry. The approach is strongly biased towards re-engineering driven software maintenance.

Two cases are described, one of which indicates maintenance requirements for CM. In this case the starting level of CM is low and the secondary role of software sets specific requirements for the CM development. The other case describes an environment where advanced techniques supporting maintenance are linked to CM features. The procedure for dealing with these cases is based on predefined requirements, scenarios to implement these requirements, and technical solutions.

KEY WORDS: software configuration management; software maintenance; application management; embedded software; embedded computer systems

1. INTRODUCTION

One of the critical issues in today's product development is how to speed up the development process without losing quality and reliability (Cooling, 1991; Salminen and Verho, 1992).

CCC 1040–550X/96/010049–27

Received 30 May 1994
Revised 14 October 1995

The situation becomes even more complicated for products whose maintenance needs may span up to 30 years in some business areas. Problems arise especially in products based on embedded computer systems, which have become more and more software-intensive. Traditional, simple version control methods are no longer sufficient because of the rate at which new product technologies are introduced. Rapidly evolving embedded software engineering environments incorporate new languages, methods and CASE tools. (VTT, 1993).

Developers of embedded software also face a highly competitive business area. Their survival depends on just-in-time delivery of customer-orientated products (Rosenthal, 1992). Embedded software is becoming influenced by additional standards such as product safety and reliability (Leveson, 1986; Redmill, 1988). This is typical, for example, in medical instruments that include product control features implemented in software (Mojdehbakhsh *et al.*, 1994).

Improvement of the software process is, for the above mentioned reasons, taking a central role in the embedded systems engineering community. Quality standard-based solutions have been developed, and many of them follow the ISO 9001 standard (ISO, 1991). Some others, especially in the US, have applied the CMM software process maturity model (Humphrey, 1988; Paulk *et al.*, 1993). The European Bootstrap model (Kuvaja *et al.*, 1994) combines both ISO 9001 and CMM model, emphasizing process improvement. In addition, such domain-specific standards as TRILLIUM for telecommunication (TRILLIUM, 1993) and the ESA standard for space applications (ESA, 1991) provide frameworks for the evaluation of the software process. In practice, however, individual organizations have to proceed by developing separate aspects of the software process step by step. Because most are still working on the 'initial' level according to the CMM model, configuration management is one of the most important concerns of the development process (Taramaa and Oivo, 1993).

The importance of software configuration management (SCM, later called CM) has been understood, but there is no clear picture on how to proceed in its improvement, in practice. One of the biggest problems is that most organizations have many valid and valuable software assets to be maintained. Almost invariably this software has been produced using several different types of methods, languages and tools. Since new software has to be continuously developed, companies need practical procedures for evolving their configuration management schemes. For this reason, we have developed an incremental approach to building CM environments and applied it in co-operation with several industrial embedded systems manufacturers. One of the key aspects of the approach is a stepwise development from simple CM practices to advanced application management methods, for the needs of embedded software development and maintenance.

2. BASIC APPLICATION MANAGEMENT ELEMENTS

There is no widely accepted definition for the concept 'application management'. It comprises activities related to managing modifications of existing applications. The definition of application management proposed in AMES (1993) is:

'Application management is the contracted responsibility for the management and execution of all activities related to the maintenance and evolution of existing applications, with well-defined service levels'.

Application management can be regarded as part of the more general architecture for the whole product life cycle including quality assurance, project management, and process model-

ling as well as application management. According to this definition application management is an extension of CM. Application management aspects have been studied and developed as part of several projects developing maintenance techniques during the last five years. These projects have been compared to each other in Boldyreff, Burd and Hather (1994).

Figure 1 indicates separate aspects which we have taken into account in CM's improvement process. The software maintenance process sets requirements for the development of CM functions, for which we use specific CM 'improvement ladders'. The application area of embedded systems sets its own requirements which favour the incremental development of the configuration and application management environment.

2.1. Starting point of configuration management

The traditional definition of configuration management is given in (Bersoff, Henderson and Siegel, 1979), according to which CM includes the following procedures (Figure 2):

- identifying and defining the configuration items for the product (configuration identification),
- controlling all changes to these items throughout the product's life-cycle (configuration control),
- recording and reporting the status of configuration items and change requests (configuration status accounting), and
- verifying the completeness and correctness of these items (configuration audit).

From the viewpoint of software maintenance, the basic role of CM is to record and create links by which a maintainer can monitor and control transition from change requests to the implementation and testing of the changes. These links include traceability, authorization, scheduling and status accounting (Zvegintsov, 1993).

Traceability forms one of the comprehensive maintenance aspects. In many cases it is separated as the only factor to be considered. Traceability can be looked at from different points of view (Nejmeh, Dickey and Wartik, 1989):

- *life-cycle relations*—dependencies between baselines produced in different phases of life-cycles,
- *evolution relations*—all version and configuration relations that keep track of the evolution of software,
- *semantic relations*—all the relations that identify the modules of code level, call-graph links between subprograms, etc.

CM has been a topic of active research since the nineteen-eighties. In the implementation of CM, advanced Unix-based solutions have played the most important role during the past few years. One of the best examples of research on comprehensive CM environments is PCTE (Boudier *et al.*, 1988), for which commercial solutions already exist (CASE, 1993). Small and Medium size Enterprises (SME) are not yet ready to use such tools for the following reasons:

- they cannot proceed directly to comprehensive CM solutions from their current CM practices,

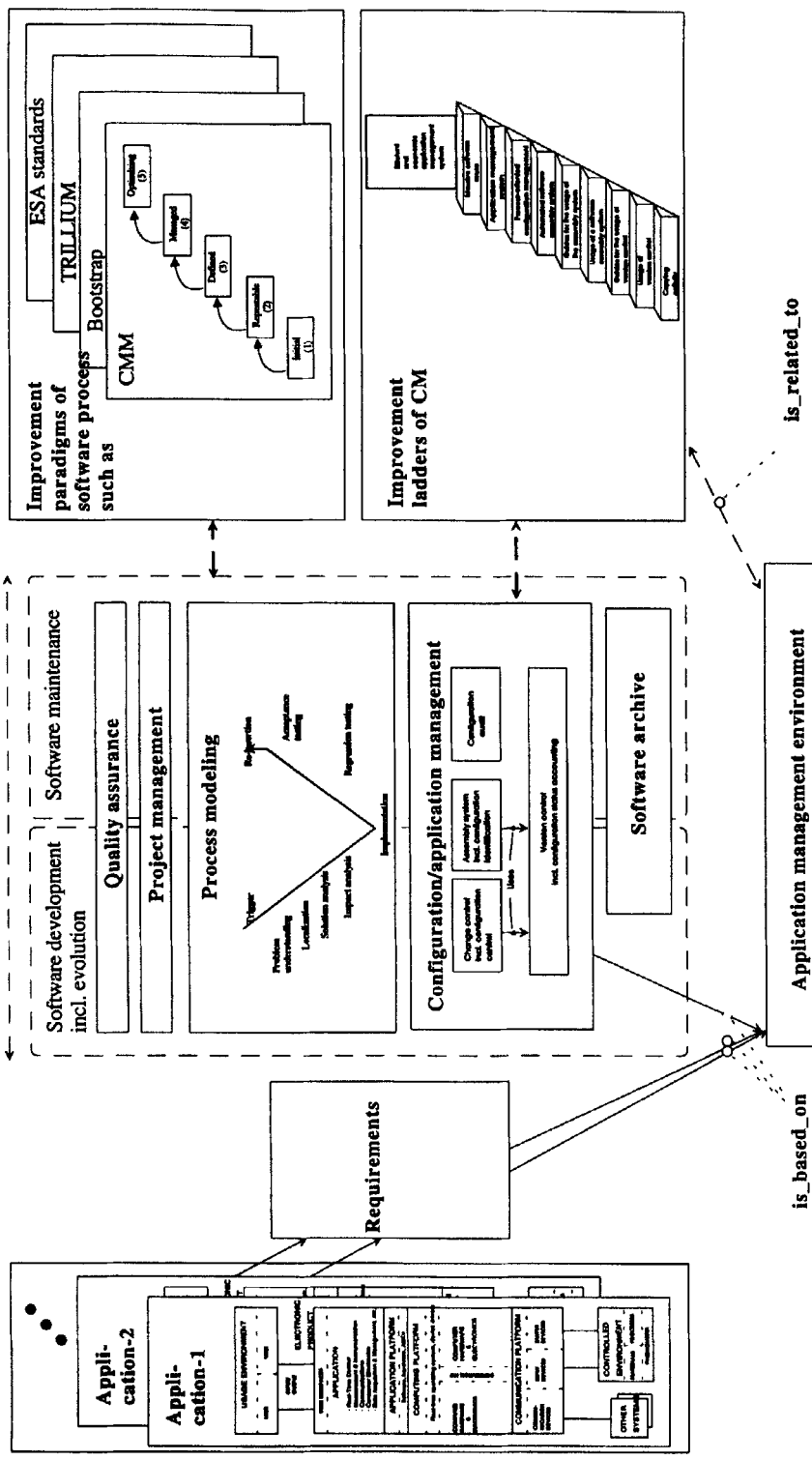

Figure 1. Basic elements for construction of application management

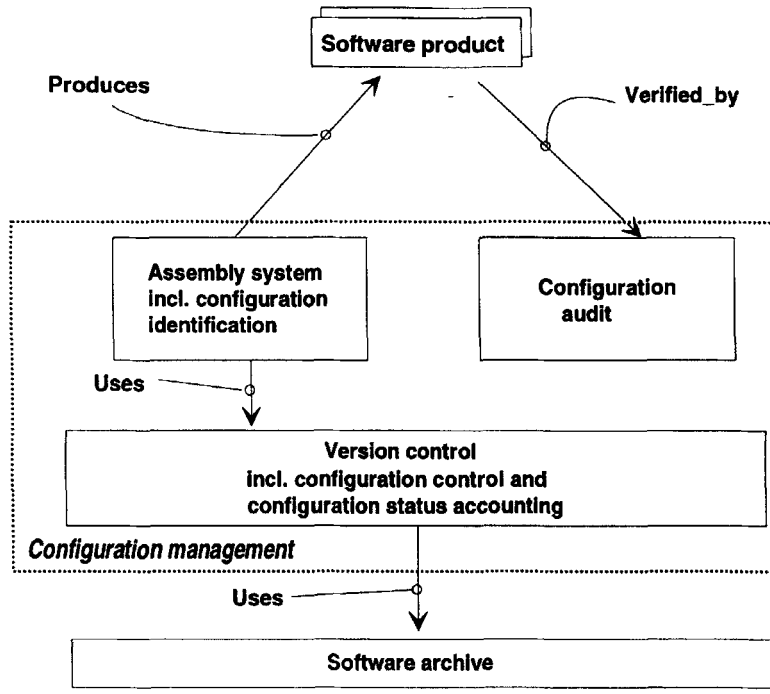

Figure 2. A traditional CM and its immediate environment

- PCTE and other similar solutions are in many cases Unix-orientated, whereas SME companies often use PC-based software development and maintenance platforms,
- the price of comprehensive CM environments is still very high.

2.2. Process modelling of software maintenance

Several researchers have proposed models for the software maintenance process (Bennett *et al.*, 1990). One of the basic principles is to outline the three main phases presented in the mid-seventies (Boehm, 1976):

- understanding the existing software,
- modification of the existing software,
- revalidation of the modified software.

A good model for the maintenance process based on this division is provided by the ESF/EPSOM project and is presented in Figure 3 (Harjani, Queille and Voidrot, 1992a). It consists of change control (the left side), implementation of change (the point of the V model), and testing of the results (the right side).

A typical trigger of the maintenance process involves information about a software problem or a request for change. The steps of the process are the following:

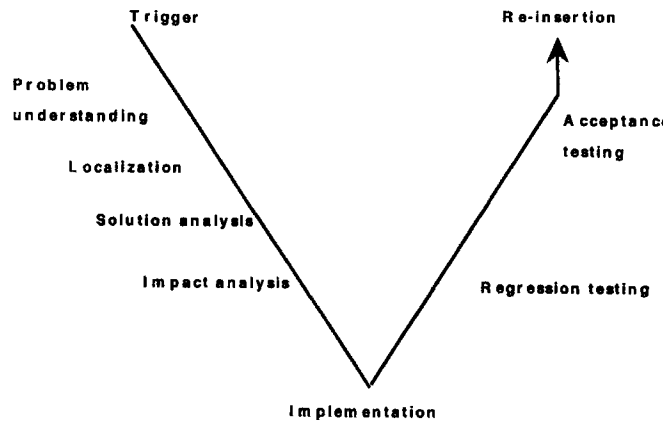

Figure 3. Software maintenance according to the ESF/EPSOM project

- *problem understanding*—the maintainer has to determine whether the problem should be attacked or not,
- *localization*—determine precisely what has to changed,
- *solution analysis*—one or several changes are devised,
- *impact analysis*—evaluate the consequences of the changes.

2.3. Maintenance challenges for embedded systems

CM is often seen to include the same aspects from one application to the another. The main differences between application areas are considered to be in the tools that are used, i.e., compilers, linkers and testing tools. The situation is, however, more complicated with embedded systems. The position of software as part of the whole product and the current state-of-the-art of CM are described below, since these factors have an effect on the further CM development of embedded software.

2.3.1. The role of software in embedded computer systems

Embedded computer programs are built-in control software for such high-value-added products as switching and production control systems, space instruments, wireless communication devices, home electronics goods, and mechatronic machines (Figure 4). Embedded software has become an important fraction of the total volume of software products (Davis, 1990). The cost of embedded software is very high in some products. For example, in telecommunication products it can constitute as much as 75 per cent of the product's development costs (Weider *et al.*, 1990). During the last few years the use of software has also increased in mechatronic products (Preston, 1993).

Responses by 27 companies to a maintenance questionnaire for Finnish firms producing embedded computer systems indicated that many companies developing embedded computer systems are in the SME category (Taramaa and Oivo, 1993). A small number of software developers in such companies are often tightly involved in everyday product development activities. Practical means for making incremental, rather than revolutionary, changes towards more effective CM schemes are thus required.

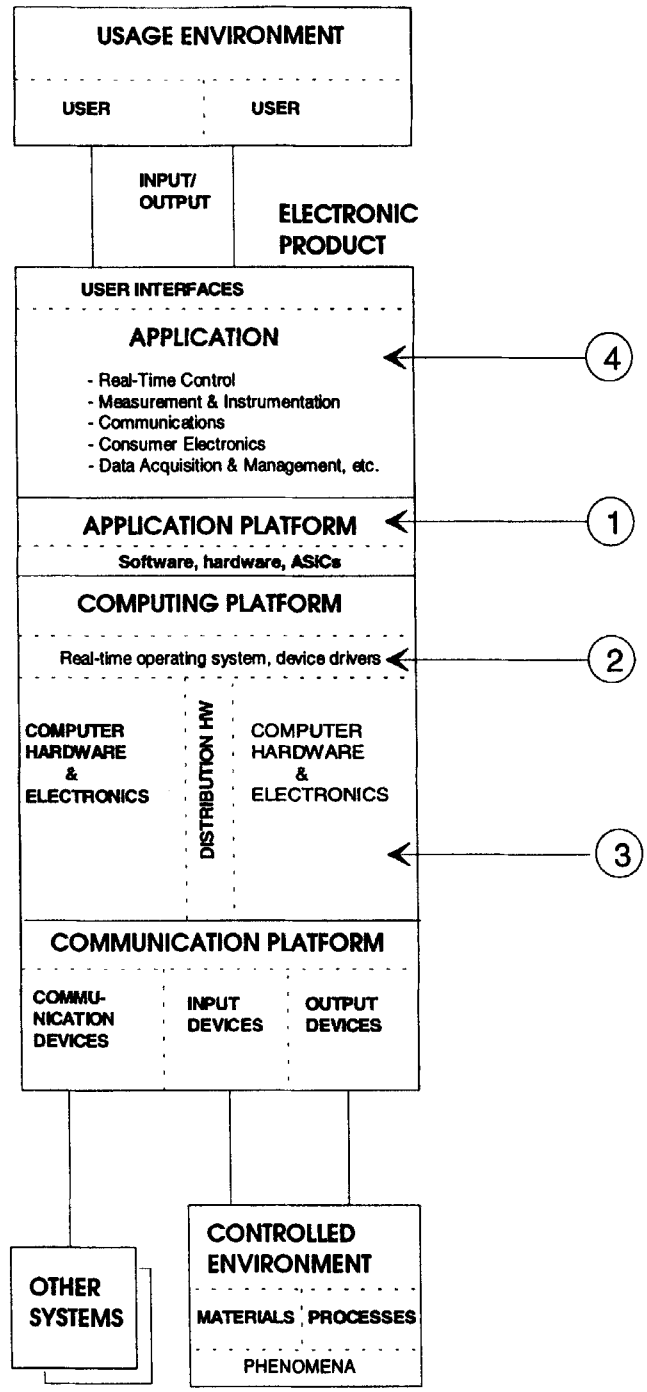

Figure 4. Software categories in embedded computer systems

In the area of embedded systems, the very first computerized CM solutions were adopted by the manufacturers of telecommunication and aerospace systems (Berlack, 1992). These applications of embedded systems were so large that CM without computerized tools was not possible.

The role of software is still variable. For example, the focus of mechatronic product developers is still not on software, which was shown by a project involving firms producing mechatronics products (Taramaa, Lintulampi and Seppänen, 1994a). This variability was confirmed by our discussions with several industrial companies (Taramaa, Seppänen and Auer, 1994b). That resulted in launching a project to evolve CM practices for industrial organizations producing embedded systems. The discussions during the planning phase of the project included around a dozen companies ranging from SME to big telecommunication firms. The existing CM practices of these companies could be categorized into three groups:

- no computerized CM,
- basic CM tools, and
- assembly mechanisms based on commercial CM tools.

Advanced solutions supporting change control will be a final category. Some computerized solutions have already been used dealing with change control. However, the discussions showed that the development of current CM practices requires a practical model that companies can apply in their own environment. Therefore, a method to assess a company's current CM level and a procedure to progress to more mature CM techniques are needed.

2.3.2. *Maintenance needs for different types of embedded software*

The general application-independent aspects of software maintenance were recognized in late seventies: *adaptive, corrective, perfective* and *preventive* (Lientz, Swanson and Tompkins, 1978). This categorization was extended by new categories presented by (Harjani and Queille, 1992b) to include:

- *user support*, where maintenance is based on an explicit request for information from the user or correcting a misunderstanding of the operation of the software,
- *evolutive* maintenance, which covers changes in functional requirements, while perfective maintenance is based on non-functional changes,
- *anticipative* maintenance, which aims at anticipating problems with the use of the software or the evolution of its running environment.

The previous maintenance categories do not deal with the structure of an application in the maintenance situation. By considering software and its separate parts, especially in embedded computer systems, different classes of software can be found which can further produce the above mentioned application-independent software maintenance needs. Embedded software falls into at least three such classes (Figure 5):

- *product software* not unique to the application (to be used in several kinds of products),
- *special system software* for the operating system, communication and device control, and
- *application software* (specific to the application).

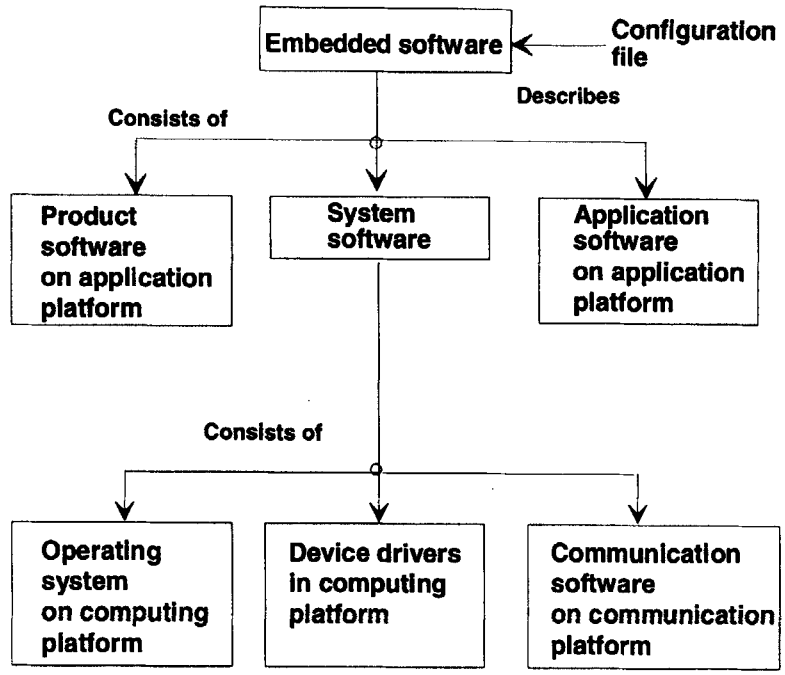

Figure 5. Typical configuration of embedded control software from the viewpoint of software maintenance

This taxonomy provides for a way to separate various maintenance concerns as follows (Figure 4):

(1) A new version of non-unique product software may be incompatible with the rest of the software.
(2) System software may be so specific that a change of operating system or hardware might require extensive modifications or a total rewrite of other code.
(3) Product software often operates only on a specific hardware platform, so that hardware changes make the original software incompatible.
(4) Application software may become incompatible with the special system software and hardware when switching to new computing or communication platforms.

3. IMPROVEMENT LADDERS OF CONFIGURATION MANAGEMENT

The spectrum of CM practices is wide; there are companies using simple PC-based solutions, while others use comprehensive development environments, where CM only forms one of the features. If developments in CM are approached incrementally, a single step forms a baseline where a company can assess the need of further work, as illustrated by the CM steps in Figure 6 (Taramaa and Seppänen, 1995). These steps have been developed and verified in several industrial projects (Taramaa and Oivo, 1993; Taramaa, Lintulampi and Seppänen, 1994a; Taramaa, Seppänen and Auer, 1994b; AMES, 1993). In Sections 3.1 and

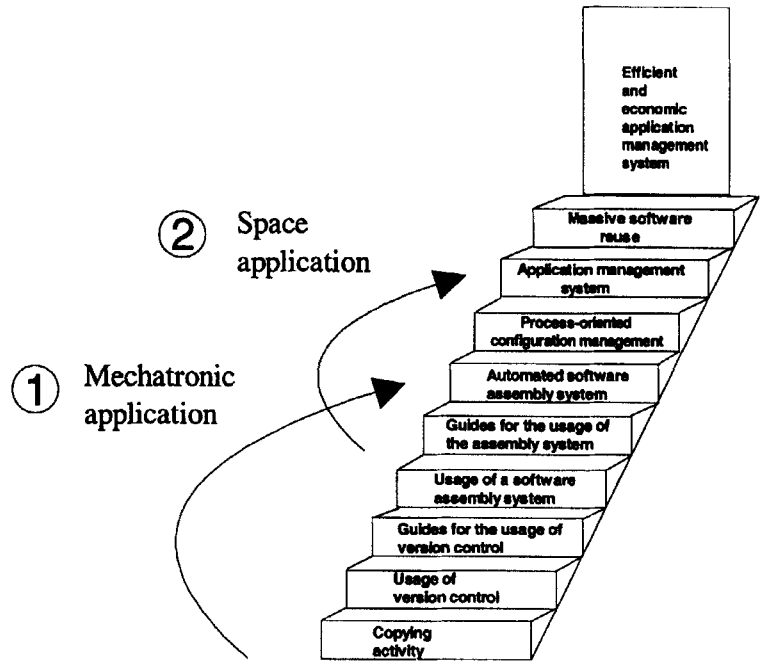

Figure 6. An incremental approach from configuration to application management for embedded software

3.2 we describe each step in more detail. Figure 6 indicates the set of steps used by each of the cases which are discussed in Sections 4 and 5.

3.1. Configuration management without explicit change management

3.1.1. Basic elements of configuration management

Copying is located at the lowest level of the CM ladder. It simply produces an archive of the delivered software. The major problems of this practice are the waste of archive space and the lack of any explicit versioning. However, the approach is viable in small organizations with few software developers and a limited range of different products.

Version control provides a means to concurrently maintain families of embedded software systems used in different types of products. One of its benefits is the saving of archive space, because versions are stored based on their incremental changes, rather than on full copies. The drawback is that the maintainer still bears the full responsibility for managing all versions. Several commercial version control systems, often based on either SCCS or RCS, are available (CASE, 1993).

Version control systems can be used in various individual ways. This will, in practice, introduce new probleems which can only be managed by making *guides for the use of the version control system.* The next example illustrates the contents of such a guide, developed for industrial embedded software development needs as part of an ISO 9000-3 based software quality handbook, e.g. presented in Table I.

At the version control step the relation to the maintenance process is weak. The changes

Table I. Part of a quality handbook dealing with version control

1. CONTENTS

2. BACKGROUND AND OBJECT OF THE DOCUMENT

This document has been produced to guide the staff of the Company NN in the usage of the PVCS version control system.

The central aspects of the project documentation are *traceability* and *versioning*. Traceability means that there is adequate information for individual recognition in each document or part of it. Versioning means that each change can be shown by a specific identification. The projects have to manage all produced versions of the documents that are disseminated.

to be made do not leave any specific trace in the maintenance process. Copying only produces a total archive copy and version control CM provides a possibility to comment on incremental changes. In the sense of traceability the only links that can be included are connections between life-cycle baselines.

A *software assembly system* consists of files, linkages between the files and compilation and linking commands stored in a configuration file. It compares the time stamps of object and source files and infers the commands to be executed for assembling the software. An assembly system remarkably speeds up the process of producing delivery-specific software packages. Examples of this principle are traditional MAKE-based solutions, pioneered on Unix systems and supported by the version control systems SCCS and RCS. A MAKE utility controls a build activity according to a script that defines dependencies between parts of assembled software packages. Table II shows examples of software assembly alternatives. On the highest level there is the application itself assembled from various binaries, such as application processes, real-time operating system modules, communication software, and initialization routines. The makefile of this level only includes a *copy* operation done in a specific order. Creating application processing modules with real-time features contains more typical makefile features such as activation of compilers, linkers, etc.

An assembly system involves the same usage problem as a version control system, i.e.,

Table II. A list of potential assembly alternatives

Directory Structure	
Application software	(*the highest level assembly* consisting of the binary to be embedded, reporting binaries, design descriptions, and other documents)
Design descriptions	(to be linked to the delivered software product)
Binary to be embedded	(*an assembly* to be created by copying different binaries, such as application processes, real-time operating system modules, communication software, and initializations)
Application processes	(*an assembly* using a makefile according to Table V consisting of application functions, communication software, real-time operating system, and hardware modules)
Documents	(e.g. the usage guides of a product, to be linked to the delivered software product)
Initializing	(*an assembly* using the conventional initializing activities; similar to application processes, but consists of specific initializing functions as well as hardware modules)
Definitions	
Source modules	
Object modules	(*an assembly* using a makefile according to Table V consisting of application functions, and the operating system for non-real-time activities)
Reporting	
Communication software	
Include modules	
Object modules	
Binary modules	(*an assembly* using the makefile according to Table V)
Hardware description modules	
Binary modules	(commercial binaries)
Real-time operating system	
Definitions	
Binary modules	
Commands	
Library modules	
Object modules	(commercial code ranging from source code to binaries)
Operating system for non-real-time activities	
Binary modules	
Library modules	
Include modules	(commercial code ranging from source code to binaries)

its everyday usage practices may become differentiated. Therefore, creating common *guies for using an assembly system* are needed. Parts of such guides can be incorporated in the user interface of the assembly system. The system can provide help menus including general and companywide instructions.

When considering traceability, the software assembly system provides the necessary conditions for constructing evolution traceability links. The record mechanism of the version-controlled software provides the possibility of tracing evolution-related links. The basic prerequisite is for version tracing provided by the formalized use of an assembly system in which each configuration can be efficiently found.

3.1.2. Automated assembly-orientated configuration management

Differences between guide-orientated assembly systems and automated software assembly systems are in the implementation of the guides. A guide-orientated assembly system is manual. In an automated assembly system the same procedures are supported by computerized tools.

A typical assembly system is strongly based on software engineering needs. In practice, this means making explicit knowledge about the software engineering tools to be used in product deliveries. *Assembly automation* provides for the hiding of such details, and thus facilitates reliability when repeating a product's assembly procedures. The functions to be managed by assembly automation include:

- generating and changing the assembled software
- automated collection of the most recent versions for an assembly
- use of version control
- tracking of customer order and product databases
- manipulating product directories
- reporting various cross references for traceability.

3.1.3. Process-orientated configuration management

During the past few years software process development has attracted increasing attention (Kellner, 1991; Dowson, 1993). Although practical software assembly systems have been developed, they do not necessarily address the software processes used by particular product manufacturers. However, implicit models of software processes can be relatively easily linked to CM systems. Configuration management tools that include implicit process models have played quite an important role when advanced CM procedures have been integrated in state-of-the-art software engineering environments. There are already some commercial CM products available that include implicit process models, such as CaseWare, ClearCase, Aide-de-Camp, and PCMS (Ingram, Burrows and Wesley, 1993).

One of the basic problems in these environments is their commitment to a fixed software process based almost entirely on the CM perspective. Flexibility demands the possibility to consider CM and software process requirements separately, although their interrelationship has to be defined. Therefore, process-orientated configuration management should be seen as consisting of two sublevels, based on either fixed or flexible process models.

Moreover, embedded systems manufacturers have to take into account product technologies

other than software. Process aspects can thus be approached from the direction of the development of either software or complete products. This will set new requirements for the second generation of commercial process-orientated CM tools.

3.2. Configuration management with efficient change management

Although CM is essential throughout the entire product life-cycle, its benefits are most obvious during maintenance (Whitgift, 1991). Software maintenance involves changing the existing software and CM is a necessary function for managing changes. It is necessary to determine which change control activities are effective for fulfilling maintenance requirements. The consideration has to deal with source code and various internal documentation since changes are more than their different combinations. In practice, this requires extensions of CM and its change control. The links between source code modules and documentation are then semantic.

Relations between the software maintenance process and CM are illustrated in Figure 7. In addition to the steps of the lower level, the linkage function is *change control* by which the assessment activities of software maintenance have been implemented. These are also called 'preparation activities' of software maintenance. The need for understanding in software maintenance has been implemented by change control. Information about the existing software is collected by the version control system and its extensions for change control and the assembly system. The software produced by the assembly system is verified by testing tools.

3.3. CM maturity model and scenarios in its implementation

Based on these steps we can present a CM maturity model as shown in Table III.

To achieve the desired CM level, the representatives of the development organization of an embedded product have to define the explicit requirements that the CM environment has to fulfill. The maintenance quality can be improved using the analysis diagram including different factors prioritized and developed, as presented in Figure 8. This analysis technique has been used in the ESF/EPSOM (Harjani, Queille and Voidrot, 1992a) and AMES (Mäkäräinen and Taramaa, 1995) projects.

The implementation of the requirements mentioned above can be done by their own procedures, which we call *scenarios*. A scenario includes a way to achieve the solution of the given requirement.

4. EXPERIENCE OF SCM IMPROVEMENT FROM COPYING ACTIVITY TO AUTOMATED ASSEMBLY-ORIENTATED CONFIGURATION CONTROL

We use maintenance systems as examples of the adoption of an automated assembly-orientated CM (Taramaa, Lintulampi and Seppänen, 1994a). Figure 9 shows a platform for an assembly system that we have built for developing and maintaining versions of mechatronic applications. It indicates the CM choices made. The starting point was the lowest level CM step, copying. Version control was added based on a commercial SCM system. The justification for this was the decisions made in similar companies concerning CM. However, the

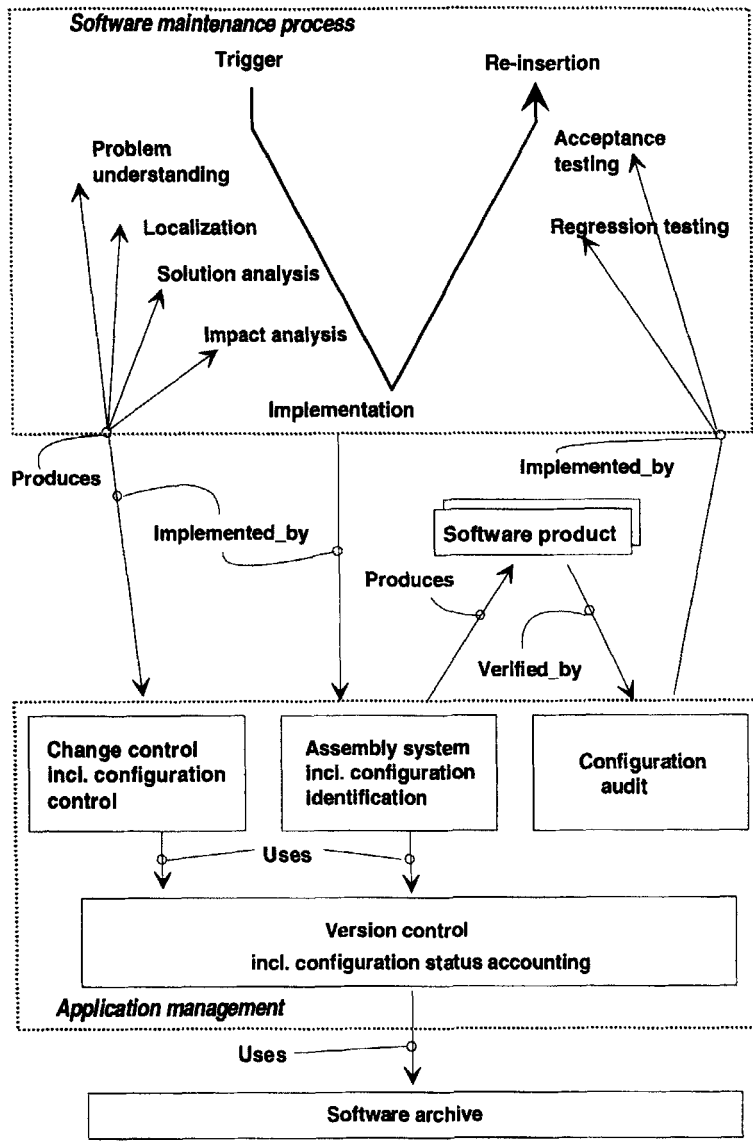

Figure 7. The relation of the software maintenance process and application management

use of CM requires making guidance for version control explicit. Version control was used, being part of the quality system of the firm, but sufficient guidance for its use was not provided.

Mechatronic products are typically used for a long time and require modernization several times. Customers expect repair and maintenance services from product manufacturers. In particular, the mechanical parts, the hardware, the sensors and the actuators of a product may have to be replaced. The software that controls such parts must then also be changed. In

Table III. Summary of the CM maturity model

Level	Solution characterization	Typical features	Typical tools
I	Copying activity	xcopy command	
II	Version control	check-in/check-out (or get/put) commands	commercial and public CM tools
III	Managed version control	writing a handbook for the use of version control system life-cycle related traces	a companywide help activity made by a text processing system
IV	Software assembly (configuration, building) system	makefiles	commercial and public builder tools
V	Managed software assembly	writing a handbook for the use of assembly system evolution-related traces	a companywide help activity made by a text processing system
VI	Automated software assembly	interactive tools and makefile definitions	a customized interactive assembly system including a tailored user interface with the ability to build specific tools and makefile definitions
VII	Process-orientated configuration management	role definitions of the CM activity	some commercial tools including tailored companywide features
VIII	Application management	computerized change control solutions semantic relation related traces	for example, the AMES environment
IX	Massive software reuse	computerized qualification of software components	companywide and research solutions

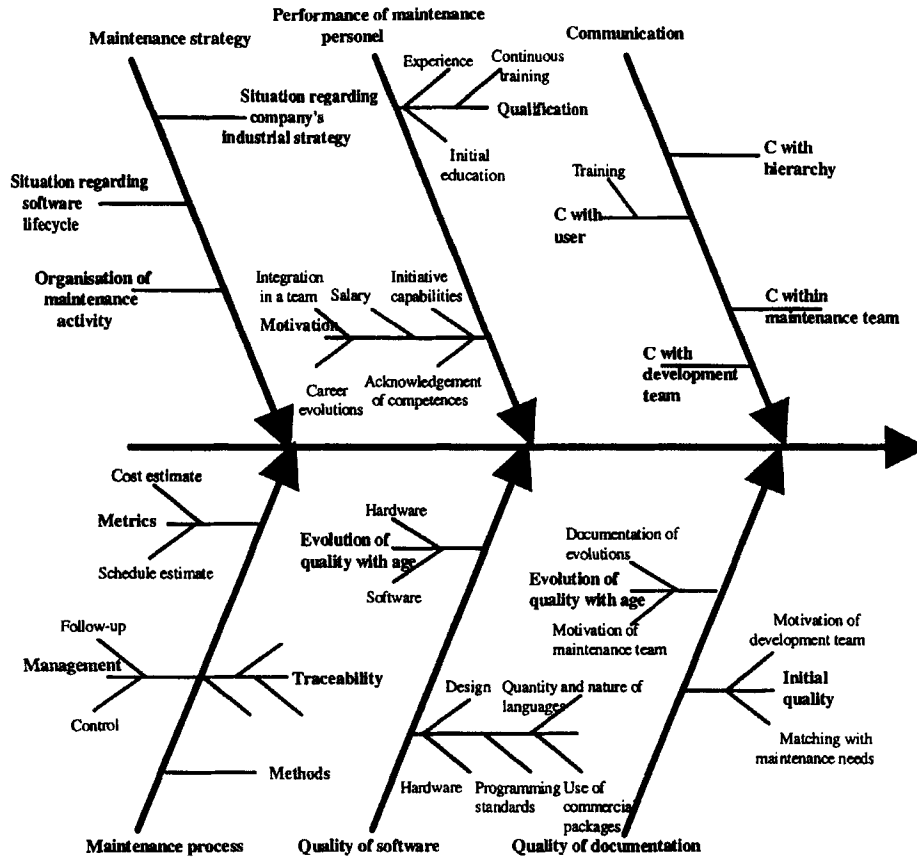

Figure 8. Main quality factors for software maintenance

addition, some software-controlled product features may need optimization and tuning to accommodate a new environment since a machine may be resold and relocated several times.

All these aspects make it necessary to trace and maintain information on the configuration of delivered machine control systems. A software assembly tool should, therefore, serve not only the initial development of a machine control system, but also its maintenance. During the initial development, the assembly tool should support incremental design, implementation and testing of the machine control features required by a particular customer. When building software packages for product deliveries, the components that implement customer-specific features should be reliably integrated with the customer-independent features and embedded into the product as a uniform control software package. In most cases, the assembly engineer is not a computer professional but a machine automation expert.

New versions of each software and hardware component may be implemented independently or as a consequence of a modification of another component. For example, if an input/output controller chip is no longer available, the hardware design of a sensor interface for a machine control system based on that chip may need to be modified. A new version of the device driver program used to control the interface would then have to be developed. The rest of the machine control software may not require any modification. The resulting

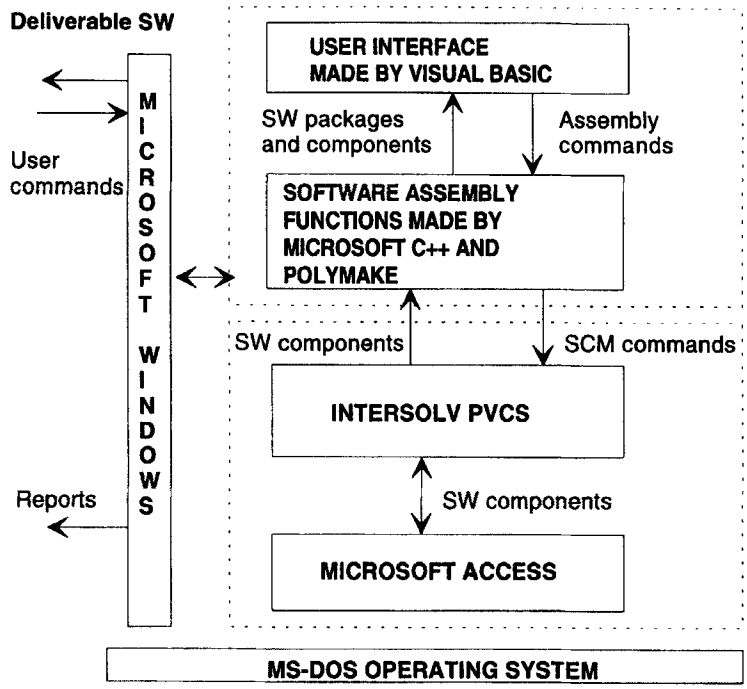

Figure 9. A prototype of an assembly system

effect is that functionally similar machine control software packages may include different software components.

Changes in the control software of a machine delivered to a customer may be made according to different maintenance policies. For example, if an electrical board of the control system is damaged, the product manufacturer may respond to the corresponding maintenance request either by:

- replacing the damaged board and its original control software package with a new copy, or
- delivering a new board that includes new device driver programs but the same application software as in the damaged board.

Some changes may be made as part of corrective maintenance initiated by the product manufacturer. For example, if a software error has been found in one or several product deliveries, a corrected version of the software may have to be delivered to the customers of those products. Other changes may involve additional machine control features required by a specific customer. A new control software package may have to be developed and delivered to that customer.

These practical concerns result in the following requirement:

RQ-LOKKI-1: All software components of machine control systems delivered to customers must be traceable.

When solving a software-related problem in a delivered machine control system, it is necessary to know the software components included in the delivery of the control system:

RQ-LOKKI-2: All software modifications to the control systems of all delivered products must be traceable.

On the other hand:

RQ-LOKKI-3: The use of specific versions of software components in product deliveries must be traceable.

Unnecessary and numerous software modifications are harmful to the image of a product manufacturer. Therefore, corrections of non-critical software errors should be done in connection with the routine upgrading of the control system:

RQ-LOKKI-4: Product-specific delivery and maintenance records of the control software are needed.

A software implementation may consist of components programmed using different languages that require specific compilers and other software development tools. Changing versions of such tools may result in upward or downward incompatibility with the tools used in the development of already delivered control software packages. For these reasons:

RQ-LOKKI-5: The software tools used in the development of a machine control software package must be available and associated with the package as long as the package will be maintained.

One of the most practical sources of information for reliable software modifications are the test cases and test results used to validate a software delivery:

RQ-LOKKI-6: Information on the unit, integration and acceptance testing of a delivered machine control software package must be maintained.

In more general terms, it is essential to store the specifications, designs and implementations produced using different tools during the software development and maintenance process in a common version management system.

We created the following scenarios for completing the requirements:

(1) Develop an automated software assembly system for producing control systems of mechatronic products. This is a fundamental requirement of the first example.
(2) Develop a reporting mechanism for traceability information which implements the requirements RQ-LOKKI-1, RQ-LOKKI-2 and RQ-LOKKI-3.
(3) Develop practices for recording information on the tools to be used in software production. This implements the requirements RQ-LOKKI-4 and RQ-LOKKI-5.
(4) Develop a makefile mechanism to take into account different product description including test cases. This implements the requirement RQ-LOKKI-6.

The first scenario was implemented by constructing a prototype of an automated machine control software assembly system. The implementation of the assembly system is based on the extension of commercial version control, database, and user interface management tools. Figure 9 depicts the computing platform of the assembly system. The system is meant to be an integrated part of the machine manufacturer's factory information system.

The platform includes a commercial version control system, such as PVCS, whose usage instructions are part of the usage guide for the platform. This is also true for the assembly tool, Intersolv's POLYMAKE, that produces the binary versions of the software to be delivered. The platform includes a specific method for the linkage between version control and assembling, implemented using Microsoft C++. Table IV presents an example of the linkage between PVCS and POLYMAKE used as part of makefiles for maintaining and producing software assemblies.

The different reports are based on a commercial database Microsoft Access implemented in Scenario 2, whilst Scenario 3 was implemented by the practices created by the R&D of a mechatronic company. Scenario 4 was based on a solution presented in Table II, where the different product documents including test cases are connected to the assembly of a product.

5. EXPERIENCES OF SCM IMPROVEMENT FROM AN ASSEMBLY SYSTEM TO CHANGE CONTROL FEATURES OF APPLICATION MANAGEMENT

The previous example discussed a solution where software assembly functions were developed as part of a CM environment for maintenance. However, this is incomplete since the assessment of change requirements includes specific items such as application and program understanding, problem understanding and management, risk and impact analysis, and configuration management. The AMES project is developing these aspects with our role being to collect requirements set by a specific application in a formal way (AMES, 1993). The implementation is based on the environment specification, collected requirements, and developed or off-the-shelf tools.

Space applications differentiate from business applications of embedded systems with regard to their maintenance period. Maintenance of space applications is more evolution orientated. Space projects take many years, but after delivery changes are almost impossible. However, changes during development are constantly being based on corrections, adaptations, and perfections of new requirements. In addition, specific features provided by the standards and recommendations of the European Space Agency (ESA) have already produced a higher CM maturity level than in many other embedded computer applications (ESA, 1991; Coene, 1992).

The requirement analysis of maintenance aspects was described by qualitative factors including the assessment of the application in general, process support, configuration management, and analysis concerning the AMES-specific elements such as application understanding, reverse engineering, impact analysis, and testing (Mäkäräinen and Taramaa, 1995). In this way we collected information for the development of the AMES environment. The requirement lists form a basis for the construction of the environment.

Below are selected items from this requirement list (AMES, 1995):

RQ-AMES-10: It should be possible to extract essential information on various documentation for the traceability platform:

Table IV. An example of a makefile-based assembly system linked with a commercial version control system

Script	Meaning
#RULES_START	
TCLIBS = c:\t_c\lib\emu c:\t_c\lib\graphics c:\t_c\lib\maths c:\t_c\lib\ch	the library references
.memswap	
.keepdir	
.arcfile .?v?	an archive extension
.path.cv_ = $(DB)	a place definition of the different archive files
.path.hv_ = $(DB)	
.path.dvc = $(DB)	
.path.tvt = $(DB)	
.cv_.c:	
get $&$<	reading operation from version control-based archive
...	
get $&$<	
%if!%defined(C-COMPILER)	a specific compiler
C-COMPILER = tcc-c-mh	default compiler
%endif	
.c.obj:	
$(C-COMPILER) $<	
product : compile_modules make_components fetch_libraries update_linking_list \	a collection of the final product
fetch_konfig_files	
tlink @linking.lst	
compile_modules: $(OBJ_MODULES)	
fetch_modules: $(MODULES)	
fetch_libraries:	
%foreach library in $(LIBRARIES)	
get-C$($(library)_DB)-R$($(library)_REV) $(library)	
%endfor	
fetch_others : $(AUX_FILES)	
fetch_konfig_files:	a description of different archive attributes
get $& gear.cfg	
get $& mkdeps.cfg	
#RULES_END	

- requirement specification documents
- functional models
- mapping between requirements and functional elements
- design models and design documents
- traceability matrices between requirements and design elements
- source code and the associated build information
- mapping between design elements and code elements
- test plans, test design, test procedures, and test results at the unit, integration, and validation levels
- verification control documents or any documentation showing which tests cases verify and which elements form the development process
- configuration management information linking versions of any kind of element to produce consistent sets of elements.

RQ-AMES-170: A tool should be provided to navigate through traceability information in an interactive way.

RQ-AMES-240: A tool should be provided to execute a query using traceability information where the selection criteria should include:

- types of entities,
- attribute values attached to entities
- existence or non-existence of any traceability relation between entities.

RQ-AMES-610: The program understanding tools should manage multitasking architectures.

RQ-AMES-680: Impact tracing should allow evaluation of impacts on the user of the software system, of changes in user manuals or operational procedures.

RQ-AMES-920: It should be possible to replace names in the existing application with more intelligible names.

RQ-AMES-1100: There should be a process management support tool for organizing and describing the activities which should be performed to change the software.

RQ-AMES-1170: Project management, application management, process support, and quality management tools should be integrated into the same platform.

These requirements were implemented by several scenarios:

(1) Create the platform for change control functions to obtain a fundamental element of an application management environment. The method for creating this platform included a lot of specific subscenarios. The implementation of this scenario included the requirement RQ-AMES-10.
(2) Adapt and assess a natural naming tool. The requirement RQ-AMES-920 was implemented by this scenario.

(3) Adapt and assess a navigation and display tool. This scenario implemented several requirements, including RQ-AMES-170 and RQ-AMES-240.
(4) Adapt and assess an impact analysis tool. This tool implemented requirements set for impact analysis, such as the requirement RQ-AMES-680.
(5) Adapt and assess problem understanding tool. Like other tools, there were several tool-specific requirements, such as RQ-AMES-610.
(6) Develop process modelling for documentation and adapted process definition. Although there were not so many requirements for process modelling, the requirement RQ-AMES-1100 implicitly included features for solving process modelling aspects.
(7) Experiment with the developed environment. It includes two subphases, tests with separate tools and then with an integrated environment. There were some requirements, such as RQ-AMES-1170, where the consistency of different support tools should be compatible with each other.

The solutions developed in the AMES project can be regarded as an extension of conventional CM. It provides the technical solutions for the software maintenance process as presented in Figure 4. A more detailed depiction is given in Figure 10. The central element is a traceability platform which makes possible the analysis of software and its different artifacts created by the development process. The tools of the AMES environment can be categorized into four groups:

- the AMES tools using the traceability platform,
- the AMES tools without treaceability information,
- the off-the-shelf tools related to the traceability platform and
- the off-the-shelf tools without any link to the AMES environment.

The AMES tools, application understanding, reverse engineering, navigation, display, and impact analysis have been related to the traceability platform (Escudie *et al.*, 1994). In addition, there are some tools which can operate without any traceability information. One of them is the disabbreviation tool InName (Laitinen *et al.*, 1995).

Traceability, platform-related, off-the-shelf tools include various configuration management tools which the industrial partners of the AMES project were already using.

6. CONCLUSION

Software development for embedded computer systems has to take into consideration different software modules made by various tools. The resources are in most cases limited. These points favour an incremental approach to developing CM. A single step forms a baseline where a company can assess the need of further work and more advanced tools. The risks of CM development can be effectively limited.

This work has described a model, the CM improvement steps, and practical solutions. The procedures to achieve a higher CM level are based on requirements and scenarios implementing the requirements. The accomplishment of scenarios forces a development organization to formalize these procedures and at the same time to make them more detailed and more practical.

At the lowest levels for using version control and assembly system, we have provided practical examples for creating guides. In addition, we have described an example of an

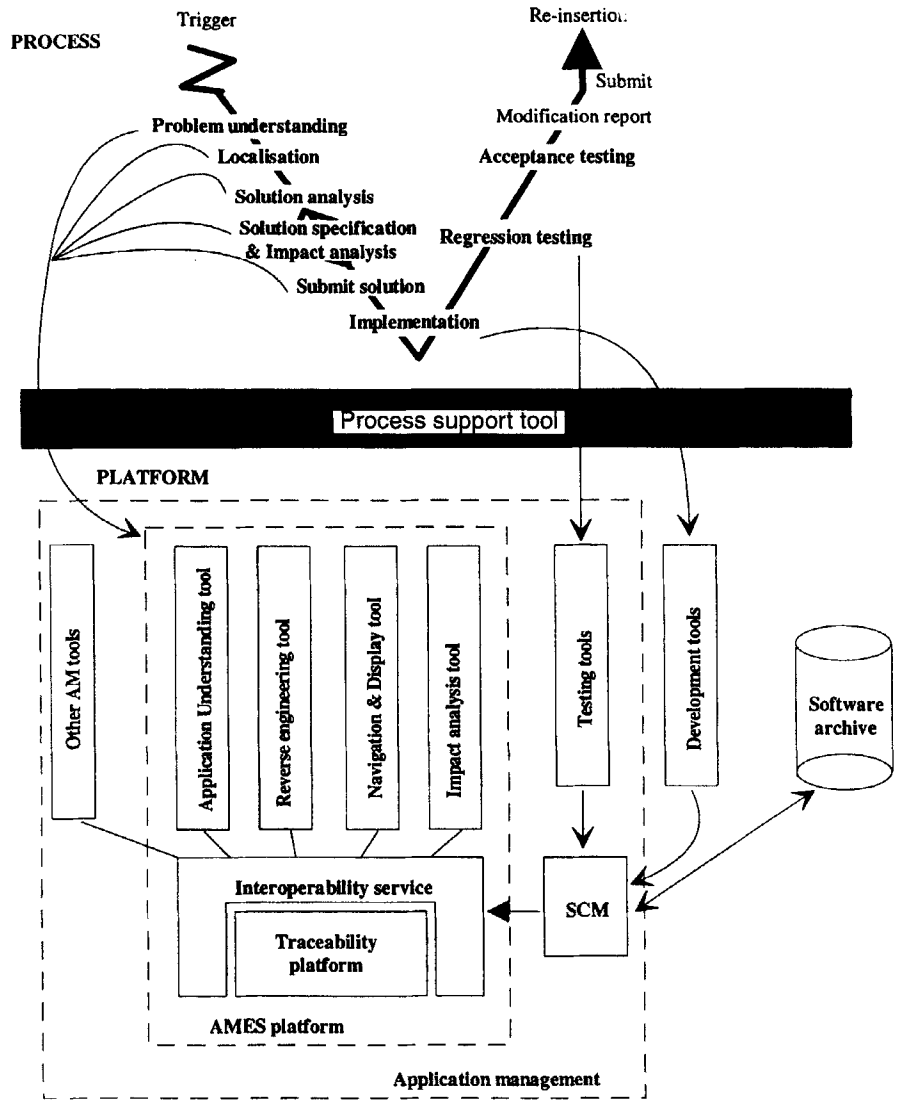

Figure 10. Relations between the maintenance process and the application management platform

automated software assembly system developed as part of the development of mechatronic products. The functions which support both change control and software maintenance set novel demands for CM. The question about whether to use CM is no longer an issue but has been replaced by issues of more comprehensive application management for maintenance and evolution of existing applications. These change control functions are not yet in use, but some of them will be implemented by the AMES project, in which VTT Electronics is one of the partners.

Achieving an automated assembly level has forced the definition of basic configuration management aspects. Advancing from configuration to application management demands the definition of the nature of the companywide maintenance process. When the company has

advanced to the application management level, it has defined its own software maintenance process. The work required to move to higher levels can be seen as a process improvement activity.

Acknowledgements

The work concerning mechatronic applications reported in this paper was funded by the Technology Development Centre of Finland (TEKES), VTT Electronics and VTT Automation. The work was done in the Finnish R&D project LOKKI in which several mechatronics firms participated. In particular, the planning of the Esprit3 project AMES (EP 8156) has resulted in new ideas concerning change control aspects. Our collaborators in the AMES project are Cap Gemini Innovation (France), Cap Programator (Sweden), Intecs Sistemi (Italy), Matra Marconi Space (France), OPL-TT (Spain), Space Systems (Finland) and the University of Durham (UK). In the final phase of writing this paper the launching of a Finnish project LEIVO, where the primary goal is to further develop existing CM practices for electronics products, provided state-of-the-art information about the industry and a wider perspective to configuration management. In addition, we would like to thank Mr. Antti Auer, Mr. Doug Foxvog and Ms. Heli Puustinen of VTT Electronics as well as Mr. Raino Lintulampi of Nokia Mobile Phones Oy, and Mr. Tommi Ketola of Space Systems Finland Ltd. for their contribution.

References

AMES (1993) *Application Management Environments and Support*, Esprit 3 Project 8156, Technical Annex, Version 1.0.

AMES (1995) *AMES Experimentation Scenarios*, Esprit 3 Project 8156, Working paper IR4.x.1, Version 1.0.

Bennett, K., Cornelius, B., Munro, M. and Robson, D. (1990) 'Software maintenance', in McDermid, J. A. (Ed), *Software Engineer's Reference Book*, Butterworth-Heinemann Ltd., Oxford, UK, Chapter 20.

Berlack, H. R. (1992) 'Software configuration management', in *Wiley Series in Software Engineering Practice*, John Wiley & Sons Inc, New York.

Bersoff, E. H., Henderson, V. D. and Siegel, S. G. (1979) *Principles of Software Configuration Management*, Prentice-Hall, Englewood Cliffs, New Jersey.

Boehm, B. W. (1976) 'Software engineering', *IEEE Transactions on Computing*, **SE-2**, 1226–1242.

Boldyreff, C., Burd, E. and Hather, R. M. (1994) 'An evaluation of the state-of-the-art for application management', in *Proceedings of the Conference on Software Maintenance*, Victoria, Canada, IEEE Computer Society Press, Los Alamitos, California, pp. 161–169.

Boudier, G., Gallo, F., Minot, R. and Thomas, I. (1989) 'An overview of PCTE and PCTE+', in *Proceedings of the Third ACM Software Engineering Symposium on Practical Software Development Environments*, Boston, MA, ACM Press, Sigplan Notices, **24**(2), pp. 107–109.

CASE (1993) *Outlook* (Special issue on Software Configuration Management ed. by G. Forte), **7**(2), 51p.

Coene, Y. (1992) *A Generic Model for ESSDE Software Configuration Management*, ESSDE Reference Facility Project ESA 8900/90/NL/US(SC), ESA/ESTEC, Noordwijk, The Netherlands.

Cooling, J. E. (1991) *Software Design for Real-time Systems*, Chapman and Hall, London.

Davis, A. M. (1990) *Software Requirements—Analysis & Specification*, Prentice Hall, Englewood Cliffs, New Jersey.

Dowson, M. (1993) 'Software process themes and issues', in *Proceedings of the 2nd International Conference on the Software Process*, Berlin, Germany, IEEE Computer Society Press, Los Alamitos, California, pp. 54–62.

ESA Software Engineering Standards (1991) *ESA-PSS-05-0, Issue 2*, ESA Board for Software Standardization and Control, Paris, France.

Escudie, A., Lambolez, P. Y., Queille, J. P., Sedes, F. and Voidrot, J. F. (1994), 'A traceability-based model for an integrated maintenance environment', in *Proceedings of the RIAO'94*, New York, Intelligent Multimedia Information Retrieval Systems and Management, pp. 358–368.

Harjani, D-R., Queille, J-P and Voidrot, J. F. (1992a) 'Maintenance in a software factory—towards an integrated maintenance support environment', in *Proceedings of the ESF Seminar*, Eureka Software Factory, Berlin, Germany, 10p.

Harjani, D-R. and Queille, J. P. (1992b) 'A process model for the maintenance of large space systems software', in *Proceedings of the Conference on Software Maintenance*, Orlando, Florida, IEEE Computer Society Press, Los Alamitos, California, pp. 127–136.

Humphrey, W. S. (1988). 'Characterizing the software process: a maturity framework, *IEEE Software*, **5**(3), 73–79.

Ingram, P., Burrows, C. and Wesley, I. (1993) 'Configuration management tools: a detailed evaluation', *OVUM report*.

ISO 9000-3 (1991) *Quality Management and Quality Assurance Standards—Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software*, ISO 9000-3:1991(E), Geneva, Switzerland.

Kellner, M. (1991) 'Software process modeling support for management planning and control', in *Proceedings of the 1st International Conference on the Software Process*, Redondo Beach, California, IEEE Computer Society, Washington DC, pp. 8–28.

Kuvaja, P., Similä, J., Kraznik, L., Bicego A., Saukkonen S. and Koch, G. (1994) *Bootstrap: Europe's Assessment Method*, Blackwell, Oxford, UK.

Laitinen, K., Taramaa, J. Heikkilä, M. and Rowe, N. C. (1997) 'Enhancing maintainability of source programs through disabbreviation', accepted for publication in the *Journal of Systems and Software*, Spring 1997, 12p.

Leveson, N. G. (1986) 'Software safety: why, what and how?', *ACM Computing Surveys*, **2**(18), 125–163.

Lientz, B., Swanson, E. B. and Tompkins, G. E. (1978) 'Characteristics of application software maintenance', *Communications of the ACM*, **21**, 466–471.

Mojdehbakhsh, R., Tsai, W., Kirani, S. and Elliott, L. (1994) 'Retrofitting software safety in an implantable medical device', *IEEE Software* **11**(1), 41–50.

Mäkäräinen, M. and Taramaa, J. (1995), 'Application management requirements for embedded software', in *Quality Management, Software Quality Management III*, Vol. 1, Ross, M., Brebbia, C. A., Staples, G. and Stapleton, J. (Eds.), *Proceedings of the Third International Conference on Software Quality Management SQM'95*, Seville, Spain, Computational Mechanics Publications, Southampton, pp. 313–320.

Nejmeh, B. A., Dickey, T. E. and Wartik, S. P. (1989) 'Traceability technology at the software productivity consortium', in *Proceedings of Information Processing 89 – IFIP*, San Francisco, California, Ritter, G.X. (Ed), Elsevier Science Publishers B.V. (North-Holland), pp. 981–988.

Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V. (1993) *Capability Maturity Model for Software*, Version 1.1, CMU/SEI-93-TR-24, Software Engineering Institute, Pittsburgh, Pennsylvania.

Preston, M. E. (1993) *Mechatronic Product Development Survey and Examples in Mechatronics*, Hewit, J. R. (Ed), CISM Courses and Lectures No. 338, pp. 19–89.

Redmill, F. J. (Ed) (1988) *Dependability of Critical Computer Systems*, Vol. 1–2, Elsevier Applied Science, London.

Rosenthal, S. R. (1992) *Effective Product Design and Development—How to Cut Lead Time and Increase Customer Satisfaction*, Business One Irwin, Homewood, Illinois.

Salminen, V. and Verho, A. (1992) 'Systematic and innovative design of a mechatronic product', *Mechatronics*, **2**(3), 257–275.

Taramaa, J. and Oivo, M. (1993) 'Evaluation of software maintenance of embedded computer systems', in *International Symposium on Engineered Software Systems*, Malvern, Pennsylvania, pp. 193–203.

Taramaa, J., Lintulampi, R. and Seppänen, V. (1994a) 'Automated assembly of machine control software', *Mechatronics*, **4**(6), 753–769.

Taramaa, J., Seppänen, V. and Auer, A. (1994b) *LEIVO—Management of Customized Versions of Embedded Software*, Project Proposal, VTT Electronics, (in Finnish).

Taramaa, J. and Seppänen, V. (1995) 'A roadmap from configuration to application management', in *Quality Management, Software Quality Management III*, Vol.1, Ross, M., Brebbia, C.A., Staples, G. and Stapleton, J. (Eds), *Proceedings of the Third International Conference on Software Quality Management*, Seville, Spain, Computational Mechanics Publications, Southampton, pp. 111–120.

TRILLIUM (1993) *Telecom Product Development Process Capability*, Version 2.3d, Bell Canada.

VTT (1993) 'Object oriented methods for embedded systems: a review', *Embedded System Engineering*, September 1993, 40–44.

Whitgift, D. (1991) *Methods and Tools for Software Configuration Management*, John Wiley & Sons, Chichester.

Weider, D., Yu, D., Smith, P. and Haung, S. T. (1990) 'Software productivity measurements', *AT&T Technical Journal*, **69**(3), 110–120.

Zvegintsov, N. (1993) 'Software configuration management: control for the software team', *Software Management News*, **11**(3), 13–24.